A05: สร้าง ALU.

ALU คือหน่วยประมวลผลที่ใช้ประมวลผลทางด้านเลขคณิตและตรรกะ. โค้ดของ VHDL ที่ให้มาในงานมอบหมายนี้เพื่อให้ศึกษา การสร้าง ALU ใน gate-level และเพื่อตรวจสอบการพฤติกรรมการทำงานของ ALU ผ่าน simulation.

VHDL เป็นภาษาที่ใช้อธิบายฮาร์ดแวร์ เพื่อที่จะใช้ศึกษาการออกแบบ hardware schematic ด้วย synthesizer รวมถึงศึกษา พฤติกรรมการทำงานของฮาร์ดแวร์ ด้วย simulator. ทั้ง synthesizer และ simulator มีผู้ผลิตอยู่มากมายรวมถึง ผู้ผลิตที่เป็น open source ด้วย.

หน้าที่

0a. ติดตั้ง VHDL synthesis and simulation tools.
คำแนะนำและอธิบายในใบมอบหมายงานนี้ ทดสอบด้วย Xilinx ISE Design Suite 13.2, ซึ่งสามารถดาวน์โหลดได้จาก: http://www.xilinx.com/support/download/index.htm.

นักเรียนสามารถเลือก synthesis and simulation tools ตัวอื่นได้. แต่ โค้ด VHDL ที่ส่งมา จะต้องสามารถทดสอบได้ด้วย Xilinx ISE Design Suite 13.2.

0b. ทำความคุ้นเคยกับ VHDL tools และ ตัวภาษา VHDL. ลองสร้าง (Synthesize) และ จำลองการทำงาน (simulate) โปรเจค CH04 ที่ให้ไว้ในภาคผนวก.

โปรเจค CH04 นี้ สร้าง one-bit ALU, ดังแสดงในรูปที่ 1 Schematic อันบน (schematic a).
ส่วน schematic อันล่าง ที่เพิ่ม overflow ขึ้นมา (schematic b) ยังไม่มีใน โปรเจค CH04 (นักเรียนจะสร้างมันขึ้นมาเอง , ดูหัวข้อถัดไป)
สำหรับ schematic a, ตัว one-bit ALU รับ `a`, `b`, `CarryIn` เป็นอินพุต, สัญญาณ `Binvert` และ `Operation` เป็นสัญญาณควบคุม, และ ให้ `Result` และ `CarryOut` ออกมาเป็นเอาต์พุต.

หน้าที่ 1. สร้าง ALU สำหรับ the most significant bit (MSB).
ตัว BitALU ที่ให้ (ดูภาคผนวก) เป็น ALU สำหรับ 1 bit ที่สามารถใช้ได้กับ bit ใดก็ได้ใน 31 least significant bits แต่ไม่ใช้เป็น MSB ไม่ได้ เพราะ ไม่มี overflow detection. ตัว one-bit ALU สำหรับ MSB ต้องสามารถที่จะบอกได้เมื่อเกิด overflow ขึ้น.

a) เมื่อไรที่ overflow จะเกิด? เติม truth table (Table 1) ให้เต็ม (a_m และ b_m, เป็น MSBs ของ สอง operands; $Binv_m$ เป็นสัญญาณควบคุมการแปลง b_m; และ sum_m เป็นผลจาก one-bit full adder ของ MSB.)
ให้ใช้ว่า overflow = 1 หมายถึง มี overflow เกิดขึ้น และ overflow = 0 หมายถึง ไม่มี overflow.
หมายเหตุ: $Binv_m$ = 1 หมายถึง operation ที่ทำเป็นการลบ (subtraction).

b) เขียน boolean expression ของสัญญาณ Overflow ในรูปของ sums of products (SOP) ของ a_m, b_m, $Binv_m$, และ sum_m.

c) สร้าง module สำหรับตรวจสอบ overflow (ด้วย VHDL). Synthesize, Simulate, และ ตรวจสอบการทำงานของมัน. (รายงานของนักเรียนควรมี code, synthesized schematic, simulation waveform, และผลสรุปการการตรวจสอบด้วย เช่น ... จาก waveform ในรูป ก. ที่เวลา t = 1us, overflow = 1 เมื่อ a_m = ... เป็นต้น)

หมายเหตุ สร้าง testbench file เพื่อทดสอบ module ที่สร้างขึ้นมา

d) รวม overflow detection module เข้าไปใน one-bit ALU เพื่อสร้าง one-bit ALU สำหรับ MSB (ดู schematic b ในรูปที่ 1). Synthesize, Simulate, and ตรวจสอบการทำงานของมัน.
(รายงานของนักเรียนควรมี code, synthesized schematic, simulation waveform, และผลสรุปการการตรวจสอบด้วย)
หมายเหตุ: นอกจาก overflow detection, ตัว one-bit ALU สำหรับ MSB ยังมีสัญญาณ set เป็นเอาต์พุตเพิ่มอีกตัว. อย่าลืมใส่มันเข้าไปด้วย.

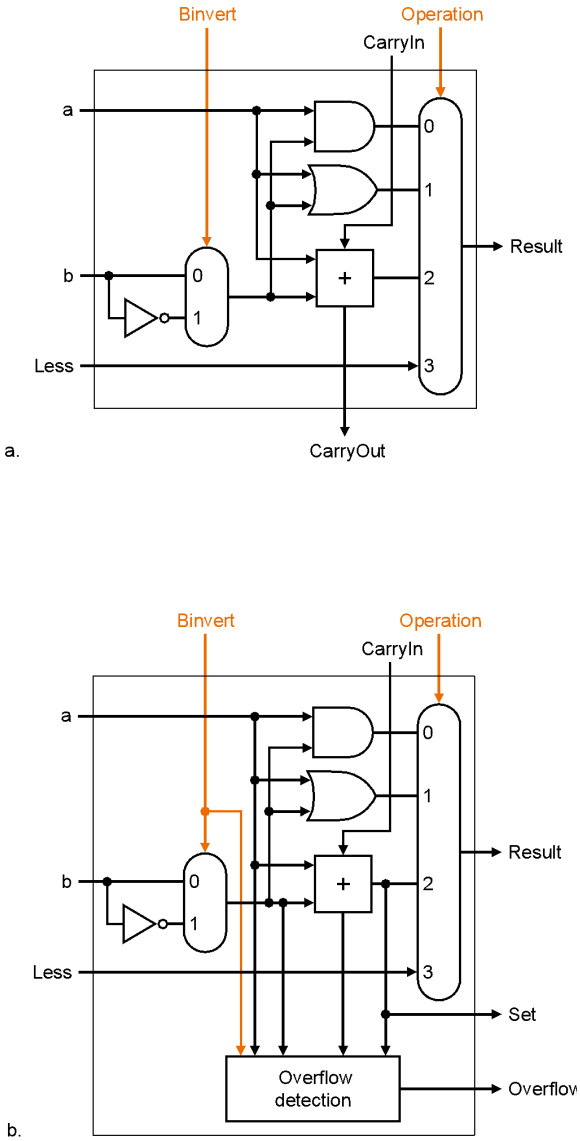

Figure 1: A one-bit ALU (from Fig. 4.17 of Patterson and Hennessy 2nd Ed. textbook)

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

FIGURE 3.3 Overflow conditions for addition and subtraction.

	$Binv_m$	a_m	b_m	sum_m	Overflow
A + B	0	0	0	0	
	0	0	0	1	
	0	0	1	0	
	0	0	1	1	
	0	1	0	0	
	0	1	0	1	
	0	1	1	0	
	0	1	1	1	
A – B	1	0	0	0	
	1	0	0	1	
	1	0	1	0	
	1	0	1	1	
	1	1	0	0	
	1	1	0	1	
	1	1	1	0	
	1	1	1	1	

Table 1: Truth table of overflow given operation (Binv), operands (a and b), and result (sum) of the MSB.

2. ประกอบ one-bit ALU ทั้ง 32 ตัวเข้าด้วยกันให้เป็น 32-bit ALU หนึ่งตัว.

ประกอบ one-bit ALU ธรรมดา 31 ตัว และ one-bit ALU สำหรับ MSB หนึ่งตัว เข้าด้วยกัน ให้เป็น 32-bit ALU หนึ่งตัว. Synthesize, Simulate, และ ตรวจสอบการทำงานของมัน.

(รายงานของนักเรียนควรมี code, synthesized schematic, simulation waveform, และผลสรุปการการตรวจสอบด้วย)

a) สัญญาณ Less และ Set มีไว้เพื่ออะไร?

อธิบายพร้อมยกตัวอย่าง.

คำใบ้: มันช่วยทำคำสั่ง set on less than (`slt`).

b) ประกอบ one-bit ALU ทั้ง 32 ตัวเข้าด้วยกัน (ด้วย VHDL) ดังรูปที่ 2.

c) เพิ่ม zero detection ดังแสดงในรูปที่ 3.

B1. * Bonus*: Improve the ALU efficiency with Carry Look-Ahead.

B2. * Bonus*: Create a multiplication module and other related modules, as necessary.

B3. * Bonus*: Create a division module and other related modules, as necessary.

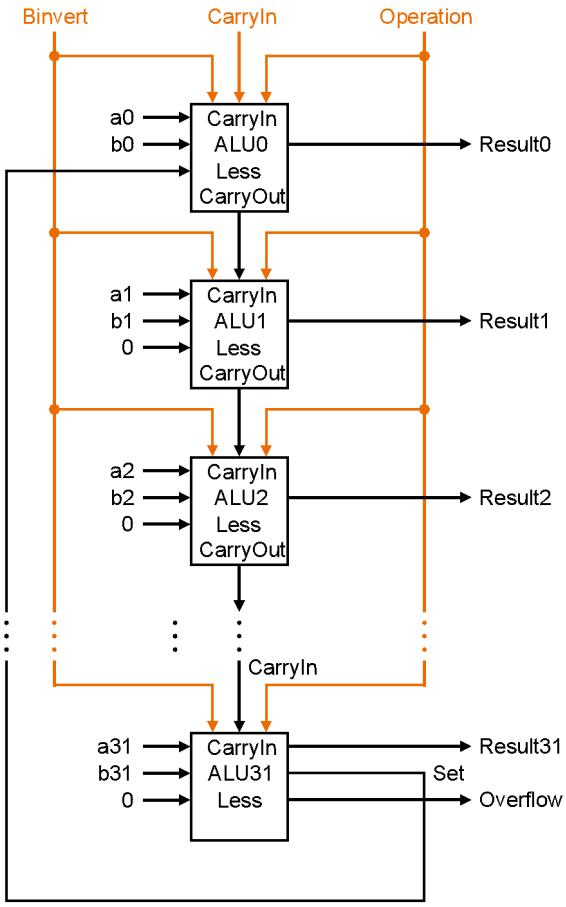

Figure 2: A 32-bit ALU constructed from 32 1-bit ALUs.

เคล็ด/เกร็ด:

บิตที่ 1 ถึง 30 สามารถทำได้แบบเดียวกัน. อาจจะเขียนต่อกันไปเรื่อยๆ หรือ, อาจจะเขียนโดยใช้ loop เพื่อสร้าง components สำหรับ 30 bits นี้ ดังโค้ดต่อไปนี้:

```
      bits1to30: for i in 1 to 30 generate
            ibit: BitALU port map(
                  a => a(i),
                  b => b(i),
                  cin => c(i),
```

```
                    Binvert => Binvert,
                    Lessin => '0',
                    Qo => qr(i),
                    cout => c(i+1),
                    operation => Operation
            );
      end generate;
```

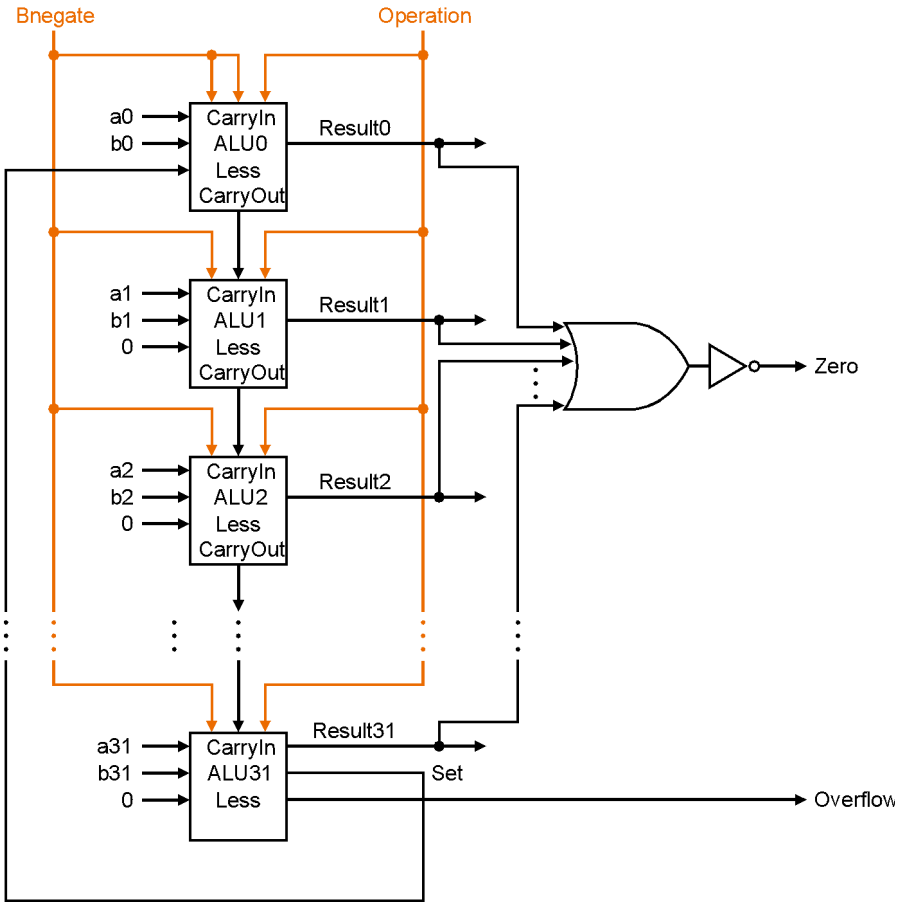

Figure 3: (Fig. 4.19 P&H 2e) ALU with zero detection

ภาคผนวก

โปรเจค CH04 สร้าง one-bit ALU, ดังแสดงในรูปที่ 1.

Source file: **BitALU.vhd**

```
----------------------------------------------------------------------------------
-- Company:  comp. engr. KKU.
-- Engineer: TK
```

```
--
-- Create Date:    08:07:26 08/18/2011
-- Design Name:
-- Module Name:    BitALU - Behavioral
-- Project Name:   CH04
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "Try not to become a man of success, but rather try to become a man of value."
-- Albert Einstein.
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BitALU is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           Qo : out  STD_LOGIC;
           cout : out  STD_LOGIC;
           operation : in STD_LOGIC_VECTOR (1 downto 0);
           Binvert : in  STD_LOGIC;
                     Lessin : in STD_LOGIC);
end BitALU;

architecture Behavioral of BitALU is

      component mux4TO1
            port ( sel : in STD_LOGIC_VECTOR(1 downto 0); in0, in1, in2, in3 : in
STD_LOGIC; Q : out STD_LOGIC);
      end component;

      component fulladder
            port (a, b, ci: in STD_LOGIC; sum, co : out STD_LOGIC);
      end component;

      signal ANDout             : STD_LOGIC;
      signal ORout              : STD_LOGIC;
```

```
      signal ADDERout    : STD_LOGIC;
      signal bi                 : STD_LOGIC;

begin
            bi <= ((not Binvert) and b ) or (Binvert and not b);

            ANDout <= a and bi;
            ORout <= a or bi;
            FA: fulladder port map (a, bi, cin, ADDERout, cout);

            MX: mux4TO1 port map (sel => operation, in0 => ANDout, in1 => ORout,
                                                                  in2 => ADDERout, in3 =>
Lessin, Q => Qo);

end Behavioral;
```

Source file: **fulladder.vhd**

```
----------------------------------------------------------------------------------
-- Company:  comp. engr. KKU.
-- Engineer: TK
--
-- Create Date:    08:19:31 08/18/2011
-- Design Name:
-- Module Name:    fulladder - Behavioral
-- Project Name:   Ch04
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "You must be the change you wish to see in the world." - Gandhi
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fulladder is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
```

```
           ci : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           co : out  STD_LOGIC);
end fulladder;

architecture Behavioral of fulladder is
begin
      sum   <= (a xor b) xor ci;
      co    <= (a and b) or (b and ci) or (a and ci);
end Behavioral;
```

Source file: **mux4TO1.vhd**

```
----------------------------------------------------------------------------------
-- Company:  comp. engr. KKU.
-- Engineer: TK
--
-- Create Date:    08:33:14 08/18/2011
-- Design Name:
-- Module Name:    mux4TO1 - Behavioral
-- Project Name:   CH04
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "Watch your thoughts; they become words. Watch your words; they become actions.
-- Watch your actions; they become habits. Watch your habits; they become character.
-- Watch your character; it becomes your destiny." -- Lao-Tze

----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux4TO1 is
    Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);
           in0 : in  STD_LOGIC;
           in1 : in  STD_LOGIC;
           in2 : in  STD_LOGIC;
```

```
           in3 : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end mux4TO1;

architecture Behavioral of mux4TO1 is

begin
   process (sel, in0, in1, in2, in3) is
   begin
      case Sel is
         when "00"  => Q <= in0;
         when "01"  => Q <= in1;
         when "10"  => Q <= in2;
         when "11"  => Q <= in3;
         when others => Q <= '0';
      end case;
   end process;

end Behavioral;
```

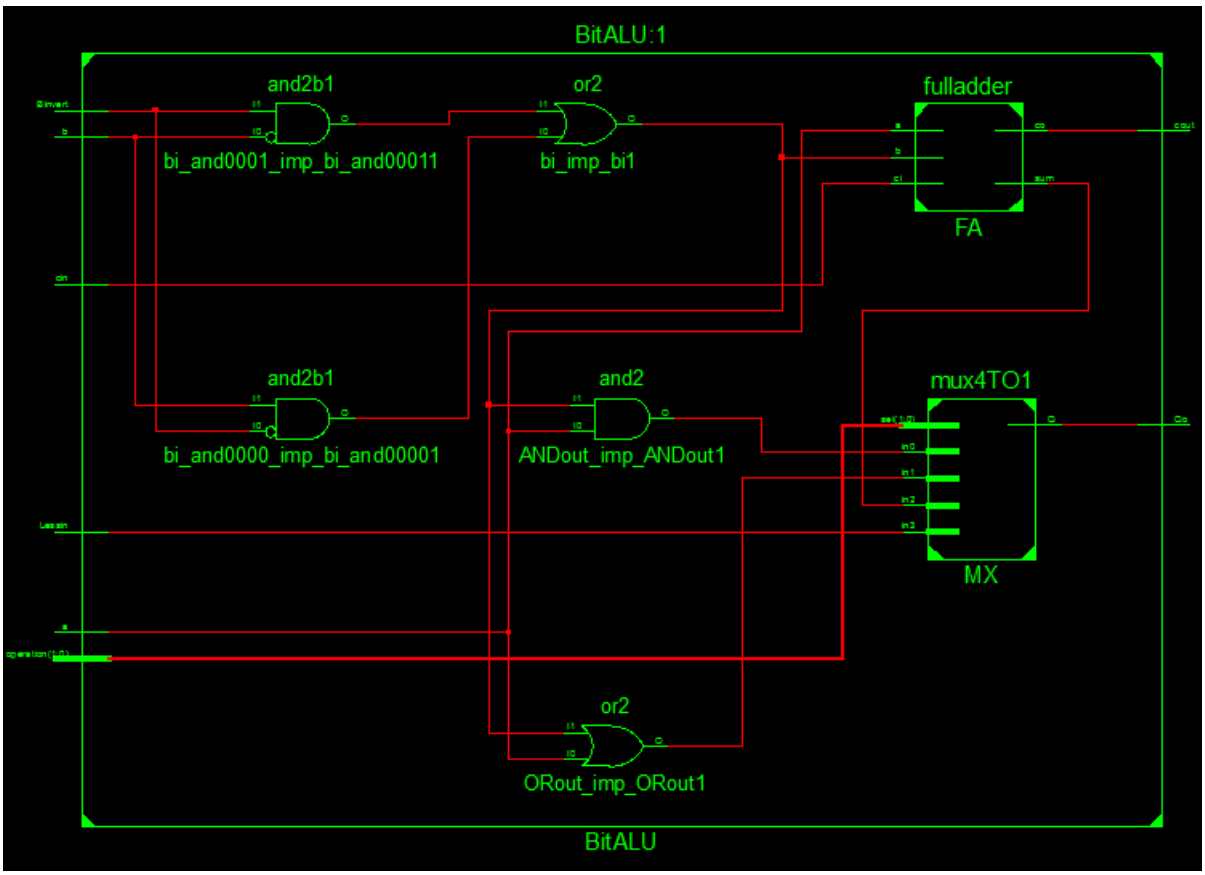

Figure 4: Synthesized RTL schematic of A 1-bit ALU.

หมายเหตุ:

ตรวจสอบว่า BitALU เป็น top module. ถ้าไม่ใช่, เลือก BitALU และ `Souce > Set as Top Module`.

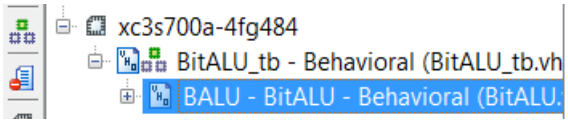

Figure 5: A test bench (BitALU_tb) is selected as a top module. Synthesis may not work.

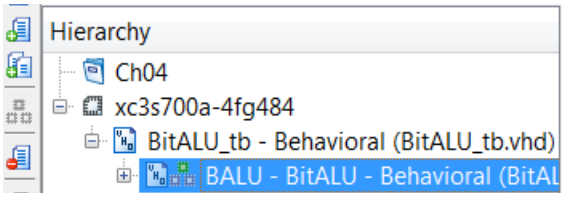

Figure 6: A BitALU is selected as the top module.

Test bench file: **BitALU_tb.vhd**

```
--------------------------------------------------------------------------------
-- Company:  comp. engr. KKU.
-- Engineer:        TK
--
-- Create Date:    13:33:08 08/18/2011
-- Design Name:
-- Module Name:    BitALU_tb - Behavioral
-- Project Name:    CH04
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- "Success is not final. Failure is not fatal. Only courage to continue that counts." -
-- Winston Churchill
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BitALU_tb is
end BitALU_tb;

architecture Behavioral of BitALU_tb is

    COMPONENT BitALU
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           Qo : out  STD_LOGIC;
           cout : out  STD_LOGIC;
           operation : in STD_LOGIC_VECTOR (1 downto 0);
           Binvert : in  STD_LOGIC;
                     Lessin : in STD_LOGIC );
    END COMPONENT;

    SIGNAL sa             : std_logic := '0';
    SIGNAL sb             : std_logic := '0';
    SIGNAL scin     : std_logic := '0';
    SIGNAL sQo      : std_logic;
    SIGNAL scout    : std_logic;
    SIGNAL sop      : std_logic_vector (1 downto 0) := "00";
    SIGNAL sBinv    : std_logic := '0';
    SIGNAL sLess    : std_logic := '0';

    constant PERIOD : time := 10 ns;

begin

        BALU : BitALU
        PORT MAP (a => sa, b => sb, cin => scin,
                  Qo => sQo, cout => scout,
                  operation => sop,
                  Binvert => sBinv, Lessin => sLess);

        PROCESS     -- running a, b, cin
        BEGIN
            RUN_LOOP : LOOP
                   sa <= '0';
                   sb <= '0';
                   scin <= '0';
               WAIT FOR PERIOD;
                   sa <= '0';
                   sb <= '0';
                   scin <= '1';
               WAIT FOR PERIOD;
                   sa <= '0';
                   sb <= '1';
                   scin <= '0';
```

```
            WAIT FOR PERIOD;
               sa <= '0';
               sb <= '1';
               scin <= '1';
            WAIT FOR PERIOD;
               sa <= '1';
               sb <= '0';
               scin <= '0';
            WAIT FOR PERIOD;
               sa <= '1';
               sb <= '0';
               scin <= '1';
            WAIT FOR PERIOD;
               sa <= '1';
               sb <= '1';
               scin <= '0';
            WAIT FOR PERIOD;
               sa <= '1';
               sb <= '1';
               scin <= '1';
            WAIT FOR PERIOD;
         END LOOP RUN_LOOP;
   END PROCESS;
```

```
          PROCESS      -- running Operation, Binv, Less
   BEGIN
                              sop          <= "00";
                              sBinv        <= '0';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "00";
                              sBinv        <= '1';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "01";
                              sBinv        <= '0';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "01";
                              sBinv        <= '1';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "10";
                              sBinv        <= '0';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "10";
                              sBinv        <= '1';
                              sLess        <= '0';
                              WAIT FOR 8*PERIOD;
                              sop          <= "11";
                              sBinv        <= '0';
                              sLess        <= '0';
```

```
                                WAIT FOR 8*PERIOD;
                                sop          <= "11";
                                sBinv        <= '0';
                                sLess        <= '1';
                                WAIT FOR 8*PERIOD;
                                sop          <= "11";
                                sBinv        <= '1';
                                sLess        <= '0';
                                WAIT FOR 8*PERIOD;
                                sop          <= "11";
                                sBinv        <= '1';
                                sLess        <= '1';
                                WAIT FOR 8*PERIOD;

            END PROCESS;

end Behavioral;
```

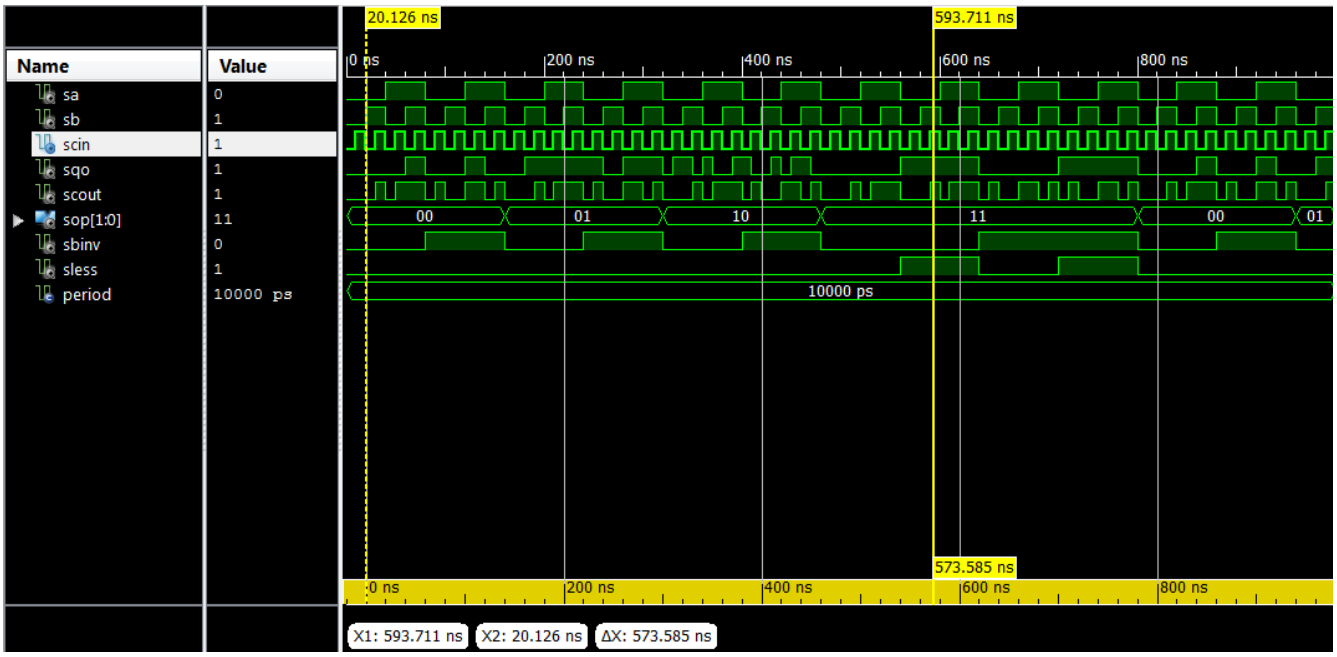

Figure 7: Simulation results (Simulation Run Time is set to 1000 ns).

Test bench: **`WordALU_tb.vhd`**

```
--------------------------------------------------------------------------------
-- Company:         comp. engr. KKU.
-- Engineer:              TK
--
-- Create Date:    09:55:02 08/22/2011
-- Design Name:
-- Module Name:    WordALU_tb - Behavioral
-- Project Name:    Ch04
-- Revision 0.01 - File Created
```

```
-- Additional Comments:
--   If you think you can do a thing or think you can't do a thing, you're right.
--   Henry Ford

----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity WordALU_tb is
end WordALU_tb;

architecture Behavioral of WordALU_tb is

component WordALU is
    Port ( a, b : in  STD_LOGIC_VECTOR (31 downto 0);
           Binvert, CarryIn : in  STD_LOGIC;
           Operation : in  STD_LOGIC_VECTOR (1 downto 0);
           Result : out  STD_LOGIC_VECTOR (31 downto 0);
           Overflow, Zero : out  STD_LOGIC);
end component;

      signal a, b : STD_LOGIC_VECTOR (31 downto 0);
      signal Binvert, CarryIn : STD_LOGIC;
      signal Operation : STD_LOGIC_VECTOR (1 downto 0);
      signal Result : STD_LOGIC_VECTOR (31 downto 0);
      signal Overflow, Zero : STD_LOGIC;

   constant PERIOD : time := 50 ns;

begin

      walu: WordALU
      port map (
            a => a,
            b => b,
            Binvert => Binvert,
            CarryIn => CarryIn,
            Operation => Operation,
            Result => Result,
            Overflow => Overflow,
            Zero => Zero);

      PROCESS
   BEGIN
```

```
            Binvert <= '0';
            CarryIn <= '0';


      -- AND --
            Operation <= "00";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)
            a <= "00111111111100000110000001101010";
            b <= "01011011110000001010010011110010";
            -- expect:  "00011011110000000010000001100010" = '0x1bc02062'
            WAIT FOR PERIOD;

      -- OR --
            Operation <= "01";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)
            a <= "00111111111100000110000001101010";
            b <= "01011011110000001010010011110010";
            -- expect:  "01111111111100001110010011111010" = '0x7ff0e4fa'
            WAIT FOR PERIOD;

      -- ADDITION --
            -- A + B; A > 0, B > 0, no overflow

            Binvert <= '0';           -- A + B
            CarryIn <= '0';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "00111111111100000110000001101010";
            b <= "00111111110000001010010011110010";
            -- expect:   "01111111011000100000101011011100" = 0x7fb1055c

            WAIT FOR PERIOD;

            -- A + B; A > 0, B > 0, overflow

            Binvert <= '0';           -- A + B
            CarryIn <= '0';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "01111111111100000110000001101010";
            b <= "00111111110000001010010011110010";
            -- expect:   "10111111011000100000101011011100" = 0xbfb1055c

            WAIT FOR PERIOD;

            -- A + B; A < 0, B < 0, no overflow

            Binvert <= '0';           -- A + B
            CarryIn <= '0';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "11000100011001010011011000000000"; -- = 0xc4653600 = -1000000000
            b <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
            -- expect: "d1000000000000111011000010100000"; -- = 0x80076140 = -2147000000
```

```
            WAIT FOR PERIOD;

            -- A + B; A > 0, B < 0

            Binvert <= '0';           -- A + B
            CarryIn <= '0';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "01000100011001010011011000000000"; -- = 0x44653600 =  1147483648
            b <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
            -- expect: "?00000000000000111011000010100000"; -- = 0x00076140 = 483648

            WAIT FOR PERIOD;

            -- A + B; A < 0, B > 0

            Binvert <= '0';           -- A + B
            CarryIn <= '0';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "10111011101000100010101101000000"; -- = 0xbba22b40 = -1147000000
            b <= "01000100011001010011011000000000"; -- = 0x44653600 =  1147483648
            -- expect: "?00000000000000111011000010100000"; -- = 0x00076140 = 483648

            WAIT FOR PERIOD;

            -- A - B; A > 0, B > 0

            Binvert <= '1';           -- A - B
            CarryIn <= '1';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "00111111111100000110000001101010"; -- = 0x3ff0606a  = 1072717930
            b <= "00111111110000001010010011110010"; -- = 0x3fc0a4f2  = 1069589746
      -- expect:   "00000000001011111011101101111000" = 0x002fbb78     =    3128184

            WAIT FOR PERIOD;

            -- A - B; A > 0, B < 0, no overflow

            Binvert <= '1';           -- A - B
            CarryIn <= '1';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "00111111111100000110000001101010"; -- = 0x3ff0606a  = 1072717930
            b <= "11111111111111101100011110000000"; -- = 0xfffec780  =     -80000
      -- expect:   "00111111111100011001100011101010" = 0x3ff198ea     = 1072797930

            WAIT FOR PERIOD;

            -- A - B; A > 0, B < 0, overflow
```

```
            Binvert <= '1';           -- A - B
            CarryIn <= '1';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "01111111111100000110000001101010"; -- = 0x7ff0606a  = 2146459754
            b <= "11111111110000101111011100000000"; -- = 0xffc2f700  =   -4000000
      -- expect:  "10000000001011010110100101101010"; = 0x802d696a     ; 2150459754

            WAIT FOR PERIOD;

            -- A - B; A < 0, B < 0

                  -- <input code for this test case>

            -- A - B; A < 0, B > 0, no overflow

                  -- <input code for this test case>

            -- A - B; A < 0, B > 0, overflow

            Binvert <= '1';           -- A - B
            CarryIn <= '1';
            Operation <= "10";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            a <= "11111111110000101111011100000000"; -- = 0xffc2f700  =    -4000000
            b <= "01111111111100000110000001101010"; -- = 0x7ff0606a  =     2146459754
      -- expect: "?01111111110100101001011010010110"; = 0x7fd29696     ; -2150459754
            WAIT FOR PERIOD;

      -- LESS (slt) --
            Operation <= "11";  -- 00: AND, 01: OR, 10: ADDITION, 11: Less (slt)

            -- a < b
            a     <= "00000000000000000000000000000101"; -- 5
            b     <= "00000000000000000000000000011100"; -- 28
            WAIT FOR PERIOD;

            -- a = b
            a     <= "00000000000000000000000000110110"; -- 54
            b     <= "00000000000000000000000000110110"; -- 54
            WAIT FOR PERIOD;

            -- a > b
            a     <= "00000000000000000000000000110110"; -- 54
            b     <= "00000000000000000000000000011100"; -- 28
            WAIT FOR PERIOD;

            -- a < b, a < 0, b < 0
            a     <= "11111111111111111111110111011111"; -- -545
            b     <= "11111111111111111111111011010100"; -- -300
            WAIT FOR PERIOD;

            -- a < b, a < 0, b > 0
```

```
            a      <= "11111111111111111111110111011111"; -- -545
            b   <= "00000000000000000000000100101100"; -- 300
            WAIT FOR PERIOD;

   END PROCESS;
end Behavioral;
```

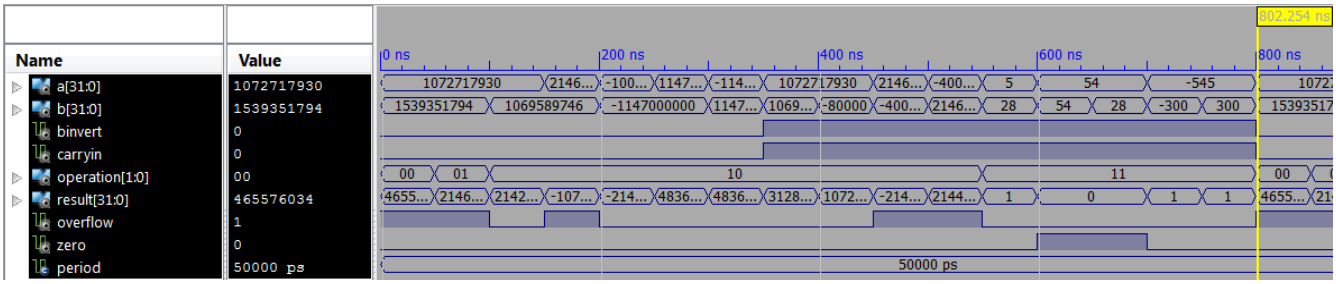

Illustration 1: Waveform captured from simulation with the given test bench (WordALU_tb.vhd)